

# 保姆级教程：个人深度学习工作站配置指南

来源 | <https://zhuanlan.zhihu.com/p/336429888>

## 1. 系统篇

### 1.1 安装Ubuntu 20.04系统

1. 在官网下载Ubuntu镜像：[Ubuntu 20.04.1 LTS](#)，得到.iso的镜像文件。
2. Windows下使用 [UltraISO](#) 制作U盘启动盘。
  - 【文件】>>>【打开】>>> 定位iso镜像文件
  - 【启动】>>>【写入硬盘镜像】>>> 选择【硬盘驱动器】，即U盘>>>【写入】
  - 【开始】>>> 等待刻录完成
3. 安装系统
  - 插入U盘>>> 开机频繁按【F2/F11/F12】(不同主板不一样)>>> 进入Boot Menu相关菜单>>> 选择U盘启动
  - 根据提示一路【enter】，语言推荐选择【english】，地区选择【shanghai】
  - 安装过程中会联网下载一些软件包更新，可以直接点skip掉。
4. 设置root账户密码：

```
sudo passwd root
```

### 1.2 配置国内镜像软件源

1. 有图形界面

**此方法选择最优镜像源**

```
打开应用程序 Software & Updates >>> Ubuntu Software >>> 定位 Download from  
>>> 点开下拉列表 >>> Other... >>> Select Best Server >>> Choose Server >>>  
enter your password >>> Close >>> Reload
```

2. 无图形界面

```
# 备份初始源
cp /etc/apt/sources.list /etc/apt/sources.list.bak

# 修改源, 粘贴镜像源进去, 如清华源, 中科大源, 阿里源, 参考[镜像源文件]一节
sudo vim /etc/apt/sources.list

# 更新
sudo apt-get update
sudo apt-get upgrade
```

### 3. 镜像源文件

更新如果出错. 需要将源中 https 改为 http

#### [中科大源](#)

```
# write to: /etc/apt/sources.list
deb https://mirrors.ustc.edu.cn/ubuntu/ focal main restricted universe
multiverse

deb https://mirrors.ustc.edu.cn/ubuntu/ focal-security main restricted
universe multiverse

deb https://mirrors.ustc.edu.cn/ubuntu/ focal-updates main restricted
universe multiverse

deb https://mirrors.ustc.edu.cn/ubuntu/ focal-backports main restricted
universe multiverse
```

#### [清华源](#)

```
# write to: /etc/apt/sources.list
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal main restricted
universe multiverse

deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-updates main
restricted universe multiverse

deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-backports main
restricted universe multiverse

deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-security main
restricted universe multiverse
```

[阿里源](#) 有需要自行访问

## 1.3 配置SSH

### 1. 安装

```
# install client
sudo apt-get install ssh

# install server
sudo apt-get install openssh-server
```

### 2. ssh 登陆

SSH 的软件架构是服务器-客户端模式（Server - Client）。在这个架构下，客户端登陆服务端。Windows下客户端需要借助 `xshell` 等软件登陆。

```
# user: 用户名
# hostname: 服务端ip地址或域名
# -p 6000: 登陆6000端口，默认22

ssh -p 6000 user@hostname
```

### 3. 免密登陆

```
# client 端操作
# create file ~/.ssh/config, write:
Host myserver
    HostName IP地址或域名
    User 用户名
    Port 端口号

# 生成密钥，一路回车
ssh-keygen

# 上传公钥到server端
ssh-copy-id myserver

# 下次登陆
ssh myserver
```

SSH 速查: [ssh cheat sheet](#)

## 1.4 frp内网穿透

### 1. 应用场景

配置一台服务器最重要的诉求，应该是可以**随时随地**去访问服务器。**从公网中访问自己的私有设备向来是一件难事儿。**

自己的主力台式机、NAS等等设备，它们可能处于路由器后，或者运营商因为IP地址短缺不给你分配公网IP地址。如果我们想直接访问到这些设备（远程桌面，远程文件，SSH等等），一般来说要通过一些转发或者P2P组网软件的帮助。

可选方案有：TeamViewer, 蒲公英, 花生壳, frp

## 2. frp是什么

简单地说, [frp](#)就是一个反向代理软件, 它体积轻量但功能很强大, 可以使处于内网或防火墙后的设备对外界提供服务, 它支持HTTP、TCP、UDP等众多协议。

## 3. 要求

- 具有公网 IP 的服务器/VPS 一台

## 4. 教程

[frp 实现内网穿透](#)

# 1.5 安装Nvidia显卡驱动

[Ubuntu 安装显卡驱动](#)

## 2. Docker安装

---

### 2.1 Docker存在的意义

1. 同一个开发团队, 不同成员, 不同操作系统之间统一环境.
2. 以下深度学习开发环境只需要在Docker中装一遍, 可以在任意设备中部署.
3. ....

### 2.2 Docker安装

[安装教程](#)

### 2.3 安装 `nvidia-container-runtime`

支持在docker容器中使用宿主GPU

[安装教程](#)

### 2.4 docker 基本操作

```
# 拉取一个镜像
docker pull ubuntu:20.04

# 列出本地镜像
docker images

## 创建并启动一个容器
# --name 指定容器名称
# --gpus 共享宿主gpu
# -v 文件夹共享
# -p 端口映射
```

```
# -i 交互模式
# -d 后台启动
# -e: 设置docker内的DISPLAY参数和宿主机一致
docker run \
    --name my_container \
    --gpus all \
    -e DISPLAY=$DISPLAY \
    -v /tmp/.X11-unix/:/tmp/.X11-unix/ \
    -v /home/liu/docker_sharing:/home/lisk/share \
    -p 20000:22
    -itd ubuntu:20.04 /bin/bash

# 查看容器
docker ps -a

# 进入容器
docker attach my_container

# 停止并退出容器
[ctrl]-d

# 启动容器
docker start my_container

# 挂起并退出容器
[ctrl]-p +[ctrl]-q

# 将容器导出为xxx.tar
docker export -o xxx.tar my_container

# 将本地文件xxx.tar导入成镜像
docker import xxx.tar my_image:tag
```

## 3. DL开发环境配置篇

本章在docker容器中进行, 同样需要修改镜像源, 根据需要配置 ssh 和 frp.

### 3.1 安装Conda环境

1. [官网下载](#)

2. 安装:

```
bash Anaconda3-2020.11-Linux-x86_64.sh
```

3. 一路 Enter + yes

4. 激活环境

```
source ~/.bashrc
```

## 5. [conda 用法](#)


### 3.1 安装CUDA

如果之前安装了旧版本的cuda和cudnn的话，需要先卸载后再安装：

```
sudo apt-get remove --purge nvidia*
```

然后按照前面的方法重新安装显卡驱动，安装好了之后开始安装CUDA：

1. 去官网下载cuda安装包：CUDA Toolkit Download | NVIDIA Developer (<https://developer.nvidia.com/cuda-toolkit-archive>)，相关选项如下（根据实际情况选择）：



1. 运行下面的命令进行安装：nvidia

```
chmod +x cuda_11.0.2_450.51.05_linux.run
sudo sh ./cuda_11.0.2_450.51.05_linux.run
```

可能会报一个警告：

```
Existing package manager installation of the driver found. It is strongly
recommended that you remove this before continuing.
Abort
Continue
```

前面已经卸载过旧版本了直接Continue就好。然后根据提示选择安装选项，注意不要勾选第一个安装显卡驱动的，因为之前已经安装过了。安装完成后提示

```

=====
Driver:   Not Selected
Toolkit:  Installed in /usr/local/cuda-11.0/
Samples:  Installed in /home/pengzhihui/SoftWares/cuda/, but missing recommended libraries

Please make sure that
- PATH includes /usr/local/cuda-11.0/bin
- LD_LIBRARY_PATH includes /usr/local/cuda-11.0/lib64, or, add /usr/local/cuda-11.0/lib64 to /etc/ld.so
.conf and run ldconfig as root

To uninstall the CUDA Toolkit, run cuda-uninstaller in /usr/local/cuda-11.0/bin

Please see CUDA_Installation_Guide_Linux.pdf in /usr/local/cuda-11.0/doc/pdf for detailed information on s
etting up CUDA.
***WARNING: Incomplete installation! This installation did not install the CUDA Driver. A driver of versio
n at least .00 is required for CUDA 11.0 functionality to work.
To install the driver using this installer, run the following command, replacing <CudaInstaller> with the
name of this run file:
    sudo <CudaInstaller>.run --silent --driver

Logfile is /var/log/cuda-installer.log
(base) pengzhihui@Fusion:~/Desktop$

```

知乎 @稚晖

2. 根据上图提示需要配置环境变量：

```
nano ~/.bashrc
```

再文件最后加入以下语句：

```

export CUDA_HOME=/usr/local/cuda-11.0
export LD_LIBRARY_PATH=${CUDA_HOME}/lib64
export PATH=${CUDA_HOME}/bin:${PATH}

```

然后使其生效：

```
source ~/.bashrc
```

3. 可以使用命令 `nvcc -V` 查看安装的版本信息：

```

(base) pengzhihui@Fusion:~/Desktop$ nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Thu_Jun_11_22:26:38_PDT_2020
Cuda compilation tools, release 11.0, V11.0.194
Build cuda_11.0_bu.TC445_37.28540450_0

```

### 3.3 安装CuDNN

进入到CuDNN的下载官网：cuDNN Download | NVIDIA Developer (<https://developer.nvidia.com/rdp/cudnn-download>)，然后点击Download开始选择下载版本，当然在下载之前还有登录，选择版本界面如下：

# cuDNN Download

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks.

☒ I Agree To the Terms of the [cuDNN Software License Agreement](#)

Note: Please refer to the [Installation Guide](#) for release prerequisites, including supported GPU architectures and compute capabilities, before downloading.

For more information, refer to the cuDNN Developer Guide, Installation Guide and Release Notes on the [Deep Learning SDK Documentation](#) web page.

Download cuDNN v8.0.5 (November 9th, 2020), for CUDA 11.1

Download cuDNN v8.0.5 (November 9th, 2020), for CUDA 11.0

Download cuDNN v8.0.5 (November 9th, 2020), for CUDA 10.2

Download cuDNN v8.0.5 (November 9th, 2020), for CUDA 10.1

Archived cuDNN Releases

知乎 @稚晖

我们选择和之前cuda版本对应的cudnn版本：

Download cuDNN v8.0.5 (November 9th, 2020), for CUDA 11.0

## Library for Windows and Linux, Ubuntu(x86\_64 & PPC architecture)

cuDNN Library for Linux (x86\_64)

cuDNN Library for Linux (PPC)

cuDNN Library for Windows (x86)

cuDNN Runtime Library for Ubuntu20.04 x86\_64 (Deb)

cuDNN Developer Library for Ubuntu20.04 x86\_64 (Deb)

cuDNN Code Samples and User Guide for Ubuntu20.04 x86\_64 (Deb)

cuDNN Runtime Library for Ubuntu18.04 x86\_64 (Deb)

cuDNN Developer Library for Ubuntu18.04 x86\_64 (Deb)

cuDNN Code Samples and User Guide for Ubuntu18.04 x86\_64 (Deb)

cuDNN Runtime Library for Ubuntu16.04 x86\_64 (Deb)

cuDNN Developer Library for Ubuntu16.04 x86\_64 (Deb)

cuDNN Code Samples and User Guide for Ubuntu16.04 x86\_64 (Deb)

知乎 @稚晖

下载之后是一个压缩包，对它进行解压，命令如下：

```
tar -xzf cudnn-11.0-linux-x64-v8.0.5.39.tgz
```

使用以下两条命令复制这些文件到CUDA目录下：

```
sudo cp cuda/lib64/* /usr/local/cuda-11.0/lib64/  
sudo cp cuda/include/* /usr/local/cuda-11.0/include/
```

拷贝完成之后，可以使用以下命令查看CUDNN的版本信息：

```
cat /usr/local/cuda/include/cudnn_version.h | grep CUDNN_MAJOR -A 2
```

可以看到版本信息如下，为 8.0.5：

```
(base) pengzhihui@Fusion:~/Desktop$ cat /usr/local/cuda/include/cudnn_version.h | grep CUDNN_MAJOR -A 2  
#define CUDNN_MAJOR 8  
#define CUDNN_MINOR 0  
#define CUDNN_PATCHLEVEL 5  
--  
#define CUDNN_VERSION (CUDNN_MAJOR * 1000 + CUDNN_MINOR * 100 + CUDNN_PATCHLEVEL)
```



## 3.6 测试

这里通过一个简单的python脚本测试一下GPU训练是否一切正常，跑一个DL里面的Hello World程序，通过两种方法测试：本地conda和docker虚拟机。

以后的开发过程中一般还是使用Docker的方式来进行更为优雅。

### 1. 本地Conda环境方式：

先用conda新建一个python3.8+pytorch1.7+cuda11.0的虚拟环境：

```
conda create --name python_38-pytorch_1.7.0 python=3.8
```

创建完成后进入环境：

```
conda activate python_38-pytorch_1.7.0
```

检查一下是否切换到所需环境了：

```
which pip
```

如果看到使用的确实是我们设置的环境目录中的pip的话说明就ok。

接下来在环境中安装pytorch，可以参考官网的安装命令：Start Locally | PyTorch (<https://pytorch.org/get-started/locally/>)

PyTorch Build	Stable (1.7.0)		Preview (Nightly)	
Your OS	Linux		Mac	Windows
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
CUDA	9.2	10.1	10.2	11.0
Run this Command:	<pre>pip install torch==1.7.0+cu110 torchvision==0.8.1+cu110 torchaudio==0.7.0 -f https://download.pytorch.org/whl/torch_stable.html</pre>			

输入以下命令进行安装：

```
pip install torch==1.7.0+cu110 torchvision==0.8.1+cu110 torchaudio==0.7.0 -f https://download.pytorch.org/whl/torch_stable.html
```

环境配置就完成了，下面新建一个简单的测试脚本验证功能，新建 `mnist_train.py`，内容如下：

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.backends.cudnn as cudnn
from torchvision import datasets, transforms
```

```

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

def train(model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))

def main():
    cudnn.benchmark = True
    torch.manual_seed(1)
    device = torch.device("cuda") if torch.cuda.is_available() else
torch.device("cpu")
    print("Using device: {}".format(device))
    kwargs = {'num_workers': 1, 'pin_memory': True}
    train_loader = torch.utils.data.DataLoader(
        datasets.MNIST('./data', train=True, download=True,
            transform=transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize((0.1307,), (0.3081,))
            ])),
        batch_size=64, shuffle=True, **kwargs)

    model = Net().to(device)
    optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)

    for epoch in range(1, 11):

```

```
train(model, device, train_loader, optimizer, epoch)

if __name__ == '__main__':
    main()
```

运行脚本，正常的话就可以看到训练输出了：

```
Train Epoch: 10 [53760/60000 (90%)] Loss: 0.128198
Train Epoch: 10 [54400/60000 (91%)] Loss: 0.117690
Train Epoch: 10 [55040/60000 (92%)] Loss: 0.153298
Train Epoch: 10 [55680/60000 (93%)] Loss: 0.116646
Train Epoch: 10 [56320/60000 (94%)] Loss: 0.113271
Train Epoch: 10 [56960/60000 (95%)] Loss: 0.147391
Train Epoch: 10 [57600/60000 (96%)] Loss: 0.107085
Train Epoch: 10 [58240/60000 (97%)] Loss: 0.114048
Train Epoch: 10 [58880/60000 (98%)] Loss: 0.216360
Train Epoch: 10 [59520/60000 (99%)] Loss: 0.253464
9920512it [03:07, 52954.13it/s]
```

知乎 @稚晖

## 2. Docker环境方式：

首先还是新建一个Docker镜像，运行下面的命令：

```
sudo docker run -it --name train_mnist \
-v /etc/timezone:/etc/timezone \
-v /etc/localtime:/etc/localtime \
-v /home/pengzhihui/workSpace/_share:/home/workspace/_share \
--gpus all nvidia/cuda:11.1-base
```

就进入到了带gpu的ubuntu20.04容器中，效果可以参考文章开头的视频。按照前面的配置方法同样配置好pytorch和其他软件包，然后运行同样的脚本，也可以得到上述输出，说明gpu在docker中正常工作。

## 4. 工作站维护篇

### 4.1 工作站系统备份还原

#### 1. 备份

由于Linux本身万物皆文件的设计理念，加上root用户对几乎全部的系统文件都有访问和更改的权限，因此Linux系统的备份和还原其实非常简单，我们直接打包整个根文件系统就可以了。

我们可以使用tar命令来打包并压缩文件系统，不过这里在打包的过程中需要排除一些不需要文件，或者与新系统文件冲突的文件，包括 /tmp、/proc、/lost+found 等目录。

找一个你想保存备份文件的目录，运行下面的命令：

```
tar -cvpzf ubuntu_backup@`date +%Y-%m-%d`.tar.gz --exclude=/proc --exclude=/tmp -
--exclude=/boot --exclude=/lost+found --exclude=/media --exclude=/mnt --
exclude=/run /
```

我们会得到一个名为 backup.tgz 的压缩文件，这个文件包含我们需要备份的系统的全部内容。

#### 2. 还原

如果系统没有出问题可以正常启动的话，那直接在刚刚的压缩包找到想还原的文件替换就好了。而如果系统无法启动了，或者说想换一块硬盘克隆一样的系统，那么可以按以下步骤操作：

- 重装干净的Ubuntu系统。跟上面介绍的一样，使用U盘给目标磁盘重装一个干净的系统，这一步是为了省去自己分配存储空间和挂载的麻烦，如果你会自己配置的话那也可以不做这一步。
- 再次使用U盘进入系统，这次选择 `try ubuntu without installing`，然后可以看到挂载好的刚刚安装了干净系统的另一个盘，我们在这里对盘里的根文件系统进行一些文件的提取：

```
sudo su

# 在tryUbuntu根目录下有media文件夹，里面是U盘文件夹和新安装的系统文件夹，在在里分别用（U盘）和（UBUNTU）表示
cd /media/（U盘）
mount -o remount rw ./

# 将新系统根目录下/boot/grub/grub.cfg文件备份到U盘中
sudo cp /media/(Ubuntu)/boot/grub/grub.cfg ./

# 将新系统根目录下/etc/fstab文件备份到U盘中，fstab是与系统开机挂载有关的文件，grub.cfg是与开机引导有关的文件，所以这一步至关重要
sudo cp /media/(UBUNTU)/etc/fstab ./

# 这一步删除新装ubuntu全部的系统文件，有用的fstab及grub.cfg已经备份
cd /media/(UBUNTU)
sudo rm -rf ./*

# 将U盘中backup.tgz复制到该目录下并解压缩
cp /media/(U盘)/backup.tgz ./
sudo tar xvpfz backup.tgz ./

# 创建打包系统时排除的文件
sudo mkdir proc lost+found mnt sys media
```

这一步完成后，在用我们在新系统中备份的 `fatab` 及 `grub.cfg` 文件去替换压缩包中解压出来的同名文件，`sudo reboot` 重启后就发现系统已经恢复到备份时的状态，包括各种框架，环境，系统设置~